W/B Planner

Solucionador para Roteamento de Veículos no Transporte Fretado de Passageiros

Alex Luciano Roesler Rese
Ciência da Computação Universidade do Vale do Itajaí
Itajaí, SC, Brasil
alexrese@univali.br

Fabricio Bortoluzzi
University of Vale do Itajaí, Brazil Noroff
University College, Norway fb@univali.br

Rafael Ballottin Martins
Ciência da Computação Universidade do Vale do Itajaí
Itajaí, SC, Brasil
ballottin@univali.br

Victor Hugo Fagundes Wachsmann
Ciência da Computação Universidade do Vale do Itajaí
Itajaí, SC, Brasil
vhwachsmann@edu.univali.br

ABSTRACT

Companies of chartered transport of passengers need to manage routes with boarding points, based on the location of the passengers. As the number of routes grow, the task of manually manage and optimize routings becomes inviable. The vehicle routing problem characterizes inside this scenario, branching into specific problems, that need different solutions for each case. Meta-heuristics are usually used to build solver algorithms for these problems. Constraint programming was chosen for helping build the solver. The main objective of the work was, to develop a system performing the solution of routings in an automatized way, by utilization of constraint programming and meta-heuristic algorithms. To minimize transport costs, travel time, and maximize the use of vehicles in an efficient and planned manner. The tests were carried out in conjunction with a routing administrator, to validate the use of the application within the research field. From the evaluations made, they demonstrated the utility of the software within its purpose. Of which it was observed that the product of this research, helped in the construction of an automated fleet consulting tool, being able to dynamically point out the distribution and operationality of a fleet of chartered vehicles in the transportation of passengers.

KEYWORDS

Pesquisa Operacional, Meta-heurística, Roteamento

1 INTRODUÇÃO

Estudado pelas últimas décadas, o problema de roteamento de veículos possui variações e abordagens que são discutidas e propostas até os tempos atuais. O cenário de um roteamento de veículos clássico, caracteriza-se com *n* veículos localizados em um depósito, dos quais devem entregar quantidades de bens a *n* clientes. Sendo os objetivos principais do problema, a otimização do custo total de transporte, redução da distância percorrida e o número de veículos [1].

Empresas de transporte fretado de passageiros, necessitam gerenciar a distribuição de seus veículos em rotas baseadas por pontos de embarque. Tais pontos são adicionados de acordo com a proximidade de cada passageiro. Após a definição das rotas com seus respectivos pontos, alocam-se veículos e motoristas que comportem a execução do roteamento em agendamentos intitulados de escalas. Entretanto, por se utilizar métodos manuais a tarefa de gerenciamento destes processos dentro de escalas diárias se torna árdua, e a análise das possibilidades de otimização entre todas variáveis e restrições muito complexa [13].

Sendo assim, o problema de roteamento de veículos é recorrente nas empresas de transporte fretado de passageiros, devido a grande demanda de transporte de funcionários e transportes especiais, como de pessoas com deficiência física [2]. Além da representatividade dentro do setor logístico, ocupando cerca de 13,8% do PIB mundial [14]. Trazendo ao cenário uma tarefa difícil de resolução manual, que na maioria dos casos acaba sendo executada de forma ineficiente por planejadores [2,3].

O problema de roteamento de veículos possui natureza NP difícil, apresenta apenas solução com pequenas instâncias e depende de heurísticas para otimizá-lo. Meta-heurísticas têm mostrado resultados promissores nas pesquisas da família de roteamento de veículos para solução com grandes instâncias. Existindo uma troca em relação a tempo e qualidade, não havendo garantia da melhor solução possível.

Utilizando tais meta-heurísticas, a literatura possui um paradigma para tais problemas restritivos como o roteamento de veículos, a programação de restrições. Auxiliando na composição de um solucionador, a programação por restrições visa um conjunto combinatório de restrições em um problema. E a partir destas restrições combinatórias, busca alcançar soluções otimizadas [4].

Encontram-se vários trabalhos dos quais abordam técnicas para a resolução das partes do problema. Porém, na pesquisa realizada não se encontrou uma solução da qual apresente um solucionador real ao problema de roteamento de veículos no transporte fretado de passageiros, gerenciando pontos de embarque e baseando-se no raio de locomoção dos passageiros.

O artigo descreve a modelagem e desenvolvimento de um sistema que automatize o processo de roteamento de veículos no transporte fretado de passageiros. Uma abordagem para tal problema, é a solução pela técnica de programação de restrições de variáveis. Portanto, a solução proposta auxiliará no processo de configuração das variáveis a serem solucionadas.

2 METODOLOGIA

O processo metodológico para aferir os objetivos propostos, foi baseado no estilo de pesquisa classificado por WAZLAWICK [5] como apresentação de algo diferente. Com o método exploratório de utilizar das ferramentas e conceitos estudados, desenvolveu-se uma solução para o problema de roteamento de veículos no transporte fretado de passageiros, validando qualitativamente a plataforma com um especialista da área de roteamentos. Elaborando por meio de abordagens técnicas de trabalhos semelhantes, uma tabela comparativa contendo artefatos de trabalhos semelhantes [5].

Trabalhos relacionados	Aplicação real	Tecnologia auxiliar	Interface gráfica
Vehicle Routing Problems: Investigação e construção de um SIG	Sim	Não	Sim
Busca Tabu para alocação de salas e professores	Parcial	Não	Parcial
Case Management Task Assignment Using OptaPlanner	Não	Sim	Sim
W/B Planner	Sim	Sim	Sim

Tabela 1: Trabalhos relacionados

A Tabela 1, lista os trabalhos estudados que foram escolhidos por conterem soluções dentro do campo de restrições de variáveis, utilizando ou não ferramentas e casos reais. Nota-se a relevância dos trabalhos com casos reais, devido a sua extensão além da academia, possibilitando uma proximidade maior com a realidade do problema proposto. Sendo estes, selecionados por meio de uma pesquisa bibliográfica da qual palavras chaves relacionadas com o problema de roteamento de veículos, programação de restrições e meta-heurísticas, descartaram trabalhos que não se encaixavam com as características traçadas.

Após elencar todos os artefatos, criou-se um artefato do qual consistiu em uma solução com características revisadas para o problema de roteamento de veículos, intitulado W/B Planner.

As entrevistas ocorreram em reuniões quinzenais, com dois especialistas administrador de roteamentos, possuindo mais de 8 anos de experiência em gerência e atendimento de roteamentos. Assim o progresso do trabalho foi apresentado, para alinhamento e ajustes. Com o objetivo de evitar problemas de modelagem ou entendimento de regras de negócio, todas as conclusões e avaliações da plataforma foram extraídas destes encontros de forma empírica e com base na experiência dos administradores.

A metodologia para o desenvolvimento utilizou-se de diagramas UML, entrevistas para levantamento de requisitos e regras de negócio, auxiliando assim na modelagem do sistema.

3 DESENVOLVIMENTO

Após a modelagem foram decididas as tecnologias que incorporaram o projeto. Desta forma o desenvolvimento conteve as seguintes características tecnológicas principais:

- *OptaPlanner*: como ferramenta base no desenvolvimento do solucionador, pela maior variedade de heurísticas, portabilidade do solucionador e opção integrada de testes de benchmark [9];
- Framework Spring: agregado com o *OptaPlanner*, para criação da API de consumo, pela compatibilidade com o *OptaPlanner*, padronização da API e ferramentas disponibilizadas pelo framework;
- React: como ferramenta base na interface de usuário web, sendo uma linguagem reativa e atual, facilitando a manutenção do ambiente e integrada com a biblioteca gerenciadora de estados Redux [10];
- Redux: para controle de estados dentro dos componentes reativos do React. Possibilitando um fluxo de estados preditivo, unidirecional, centralizado, testável e flexível. Gerando um desacoplamento entre a camada de negócios e a interface do usuário, além de proporcionar uma facilidade para colaborações no futuro da pesquisa, pela forte padronização e disseminação da linguagem na comunidade [11].
- Github: Controlador de versão do código-fonte da aplicação sendo de código aberto e integrando ao Heroku;
- *Heroku*: Como serviço para distribuição na web da ferramenta, viabilizando os testes remotamente e sem necessidade de uma estrutura física para servir o sistema;

O fluxograma principal do sistema pode ser visualizado na Figura 1, da qual exibe o processo de criação e execução do solucionador, contendo todos os passos necessários de acesso e configuração para geração das soluções.

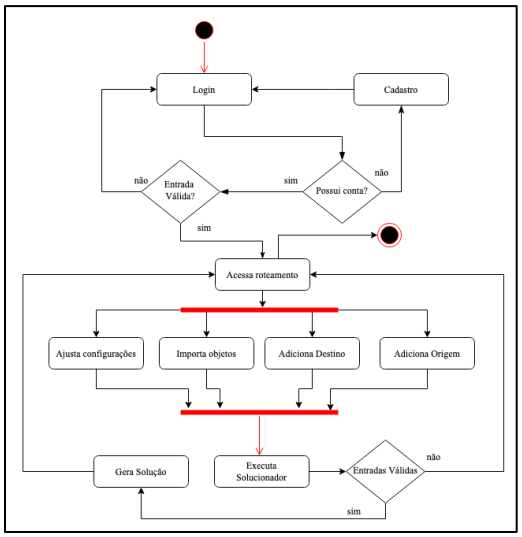

Figura 1: Fluxograma principal do sistema

A programação de restrições foi o paradigma escolhido para ser abordado neste projeto. Seguindo os princípios deste paradigma, a primeira etapa de modelagem de um problema de satisfação de restrições, trata da definição das variáveis do problema de satisfação de restrições, sendo definidas como rígidas e suaves. As variáveis rígidas não devem ser quebradas, e as suaves se possível devem ser acatadas.

As variáveis elencadas para o domínio do problema com suas respectivas restrições rígidas são as seguintes:

- Passageiros;
    a. Passageiros necessitam de um ponto de embarque dentro do raio configurado;
    b. Todos os passageiros devem ser atendidos;
- Capacidade de Veículo;
    a. Veículos não devem exceder a capacidade respectiva;
    b. Todo veículo deve conter dois lugares vagos para imprevistos;
- Motorista;
    a. Um motorista não pode realizar mais de uma rota no mesmo período.

Desta forma, as variáveis elencadas com suas respectivas restrições suaves são as seguintes:

- Distância de Rota;
    a. Deve se priorizar rotas com menor distância;
- Tempo de viagem
    a. O tempo de uma viagem não deve exceder o limite configurado;
- Motorista
    a. Alguns motoristas são fixos ou preferem certos turnos de trabalho.

Observa-se que as restrições que adicionam gerência de tempo no problema, não foram aplicadas na modelagem que será apresentada, por não fazerem parte do objetivo principal do trabalho.

3.1 Especificações de software

As especificações de software elencadas para este projeto foram compostas de requisitos funcionais e não funcionais, regras de negócios e diagramas UML.

Os requisitos funcionais para o sistema, focaram-se na criação e autenticação dos planejadores, roteamentos, configurações e importações relacionados aos roteamentos. Além das restrições já comentadas como raio de embarque, importação de passageiros, veículos e pontos de embarques automáticos.

Os requisitos não funcionais focaram-se na utilização das ferramentas de desenvolvimento como o OptaPlanner, sendo o solucionador e a biblioteca Drools como modelador de restrições. Integrados ao Spring Boot por uma API RESTful, utilizando do GraphHopper como servidor de direções para rotas precisas. O banco de dados escolhido para o projeto foi o Postgres, por ser um SGBD de código aberto, relacional, com mais de 30 anos de desenvolvimento, do qual possui confiabilidade, robustez e performance [6]. Para camada de interface de usuário se requere a utilização do React, juntamente com o Redux, além da incorporação do Google Maps para exibição dos roteamentos. Tal infraestrutura pode ser visualizada na Figura 2, da qual exibe o diagrama de componentes construído para o sistema.

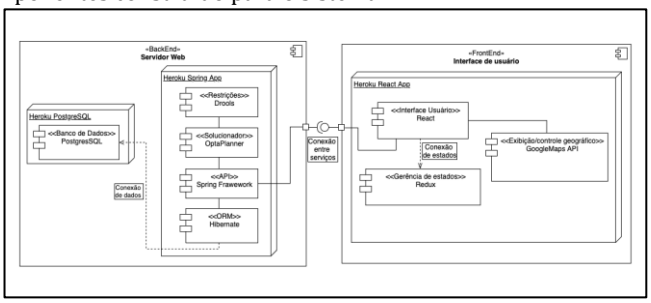

Figura 2: Diagrama de componentes

Sendo as regras de negócio baseadas principalmente nas restrições já comentadas, além da autenticação de cada planejador, o sistema deve fornecer as informações de solução do roteamento configurado, necessitando a visualização de cada rota, respectivos pontos de embarque, passageiros atrelados, distância, veículos e capacidade.

3.1.1 Diagramas UML

Como é exibido na Figura 3, o diagrama de casos de uso contém ações necessárias para o administrador de roteamentos como usuário, criar e configurar os roteamentos que serão solucionados. Além das funções autônomas disparadas por algumas ações específicas do usuário, como a distribuição de

pontos de embarque de passageiros de acordo com passageiros importados.

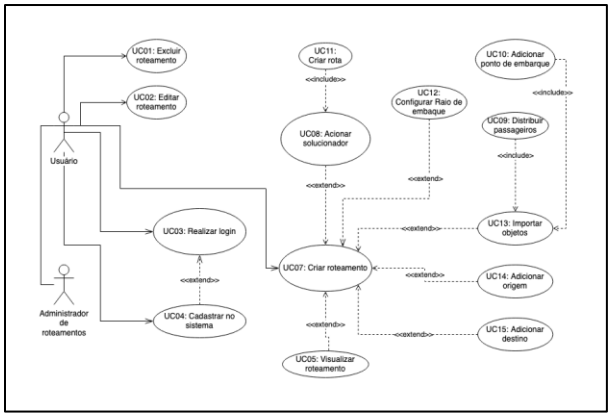

Figura 3: Diagrama de casos de uso

Após a modelagem dos casos de usos, foram elencadas as principais classes de domínio do problema. A Figura 4 exibe o diagrama final contemplando o problema de roteamentos. Nota-se a relevância da classe *Route*, da qual conectada com a classe *Routing*, possui as informações das rotas criadas pelo solucionador, a partir das configurações e objetos adicionados, como *Passenger, Vehicle e RangePoint*.

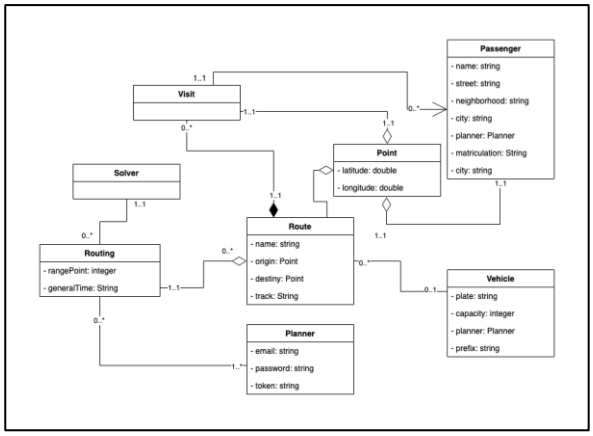

Figura 4: Diagrama de classes

Juntamente com o diagrama de classes, construiu-se um diagrama para expressão das classes do solucionador como consta na Figura 5. Sendo estas compostas de parâmetros e configurações necessárias para o processamento feito pelo *OptaPlanner*.

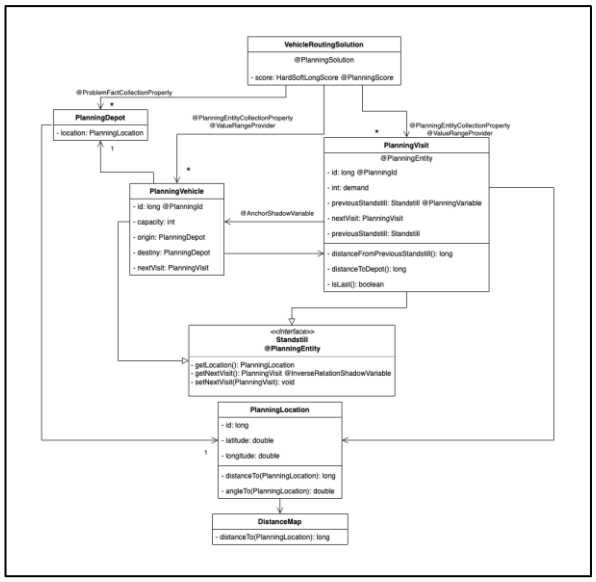

Figura 5: Diagrama de classes para o solucionador

A classe que comporta a solução é a *VehicleRoutingSolution*, contendo visitas feitas por cada veículo, combinadas com suas rotas, distância e *score* da solução. O tempo total é calculado pela classe auxiliar *DistanceMap*, conectada a classe *PlanningLocation* que obtém o tempo total em milissegundos do trajeto. A classe *PlanningVisit* utiliza o conceito de encadeamento, do qual o atributo *nextVisit* aponta para uma referência da mesma classe, desencadeando o fluxo de visitas para a solução. A classe *PlanningVehicle* é incorporada dentro da classe *PlanningVisit* com a marcação @AnchorShadowVariable, possibilitando ao veículo o retorno do sequenciamento de cada visita ordenada pelo encadeamento e direcionada ao mesmo.

As conexões com o *OptaPlanner* para definição do problema são feitas pelos JavaBeans anotados na Figura 5, contendo para esta aplicação as seguintes:

- *@PlanningSolution*: Representa o problema e uma possível solução que é reconhecida e utilizada pelo *OptaPlanner* para o processamento. Geralmente adicionada na classe principal do problema, neste caso a *VehicleRoutingSolution*;
- *@PlanningEntityCollectionProperty*: Para extrair as instâncias da solução o *OptaPlanner* precisa recorrer a classe demarcada como *@PlanningSolution*, desta forma as entidades ou *getters* demarcados com este *Bean* serão utilizadas no processamento das soluções. No diagrama da Figura 5, pode-se notar as classes *PlanningVehicle* e *PlanningVisit* sendo demarcadas com o *Bean*, que necessitam ser modificadas e manejadas durante o processamento da solução;
- *@PlanningScore*: Uma classe *@PlanningSolution*, obrigatoriamente necessita de uma variável demarcada com *@PlanningScore*. Desta forma quando executado, o

OptaPlanner utiliza da variável para atribuir as pontuações em torno da solução, adicionando as configurações feitas, pontos negativos ou positivos em relação as restrições aplicadas;

- *@ProblemFactCollectionProperty*: Dentro de uma classe *@PlanningSolution*, também se necessita uma ou mais variáveis *@ProblemFactCollectionProperty*. Tais quais são utilizadas como fatos base para as soluções, como por exemplo o *@PlanningDepot*, que representam a origem ou destino do problema, sendo fatos fixos que participam da solução. Carregados geralmente uma vez por execução e servindo tanto para o cálculo das soluções quanto para os scores;
- *@PlanningEntity*: Demarcação feita para sinalizar ao *OptaPlanner* a utilização de tal classe dentro das soluções;
- *@PlanningVariable*: Necessária dentro de uma classe *@PlanningEntity* para demarcação das variáveis utilizadas na solução. Necessitando também de pelo menos uma propriedade *@ValueRangeProvider*, referenciada pelo nome a uma outra variável dentro da classe @PlanningSolution.
- *@PlanningId*: Necessário para utilização do solucionador em *multi-thread* e resultados em tempo real, como foi implementado neste projeto. Desta forma o *OptaPlanner* mapeia entidades e variáveis entre threads, controlando os estados da solução;
- *@ValueRangeProvider*: Define possíveis valores da variável demarcada, referenciando um *@ValueRange* dentro da classe principal demarcada com o *@PlanningSolution*.
- *@AnchorShadowVariable*: Demarca a âncora de uma variável encadeada apontando o nome da variável referente;
- *@InverseRelationShadowVariable*: Define e conecta a variável demarcada a uma variável *@AnchorShadowVariable*. No diagrama da Figura 5 o atributo *nextVisit* da classe *PlanningVehicle*, conecta-se por este *Bean* com o atributo *vehicle* dentro da classe *PlanningVisit*. Tal relacionamento permite ao *OptaPlanner*, reconhecer essa cadeia de eventos e retornar as sequências corretas configuradas pelo solucionador [9].

As configurações necessárias para serem aplicadas ao solucionador, demandaram um estudo aprofundado das mecânicas do *OptaPlanner*. A necessidade da inserção de um serviço de direções, dificultaram a implementação, devido principalmente a complexidade de integração de todas as tecnologias. Uma dificuldade desta etapa foi o entendimento e posteriormente a aplicação das configurações de modelagem do *OptaPlanner*, para servir como solucionador. Além da utilização imprevista do *GraphHopper*, para consulta das informações de roteamento pelo solucionador, do qual a documentação completa e exemplificada da biblioteca auxiliou na recuperação do tempo no escopo de desenvolvimento.

3.2 W/B Planner

O sistema proposto como foi visto na modelagem foi dividido em dois serviços principais, o solucionador e a interface de usuário como aplicação web. O código fonte da aplicação pode ser visto integralmente no repositório https://github.com/wachsmann/wbp-frontend e https://github.com/wachsmann/wbp-back.

3.2.1 Solucionador

Como visto na Figura 5, o solucionador contém suas próprias classes configuradas para execução do *OptaPlanner*. Em conjunto com as configurações foi necessário a implementação da biblioteca de serviços de direções *GraphHopper*, para proporcionar o retorno das direções entre pontos, fazendo com que o solucionador tenha uma acurácia mais realista, além de retornar as rotas sugeridas para serem transpostas no mapa. Retirando assim do serviço os dados de distância, conectando-se ao serviço embarcado do *GraphHopper* para encontrar as menores rotas, obedecendo e otimizando a capacidade de atendimento do roteamento.

```
1   Constraint vehicleCapacity(ConstraintFactory constraintFactory) {
2     return constraintFactory.from(PlanningVisit.class)
3       .groupBy(
4         PlanningVisit::getVehicle,sum(PlanningVisit::getDemand))
5       .filter(
6       (vehicle,demand)-> demand > (vehicle.getCapacity()-2))
6       .penalizeLong(
7           "vehicle capacity",
8           HardSoftLongScore.ONE_HARD,
9           (vehicle,demand) -> demand - (vehicle.getCapacity()-2)
10      );
11    }
12  Constraint distanceFromPreviousStandstill(
13    ConstraintFactory constraintFactory){
14      return constraintFactory.from(PlanningVisit.class)
15        .penalizeLong(
17          "distance from previous standstill",
18          HardSoftLongScore.ONE_SOFT,
19          PlanningVisit::distanceFromPreviousStandstill
20        );
21    }
22  Constraint distanceFromLastVisitToDepot(
23    ConstraintFactory constraintFactory) {
24      return constraintFactory.from(PlanningVisit.class)
25        .filter(PlanningVisit::isLast)
26        .penalizeLong(
27          "distance from last visit to depot",
28          HardSoftLongScore.ONE_SOFT,
29          PlanningVisit::distanceToDepot
30        );
31  }
32
```

Quadro 1: Código-fonte - restrições aplicadas ao solucionador

As restrições foram adicionadas ao planejador através da biblioteca score stream, que abstrai o Drools dentro do Java, não necessitando a utilização de arquivos para importação das restrições. O código implementado pode ser visualizado no Quadro 1. Foram adicionadas três restrições para processamento no OptaPlanner, vehicleCapacity, distanceFromPrevisouStandstill, distanceFromLastVisitToDepot. Todos os métodos utilizam a classe ConstraintFactory para suas criações e aplicações dentro do solucionador. O método vehicleCapacity com base na classe PlanningVisit, agrupa veículos com as demandas somadas de cada visita a um ponto de embarque. Filtrando as combinações que se encaixam dentro da restrição de capacidade do veículo. Assim adiciona penalizações caso a demanda ultrapasse a capacidade do veículo. Desta forma, configura ao score uma pontuação negativa e rígida pela marcação HardSoftLongScore.ONE_HARD. Nota-se que a regra de todo veículo possuir 2 lugares vagos, foi adicionada ao filtro quando a comparação entre a demanda e a capacidade do veículo é expressa. Os métodos distanceFromPreviousStandstill e distanceFromLastVisit, calculam a distância entre dois pontos (o anterior com o atual), utilizando o método dentro da classe de planejamento que retorna o valor da distância. Sendo estas penalizações suaves que diferem entre si pelas suas aplicações. A primeira trata as distâncias das visitas feitas pelos veículos, e a segunda da distância entre os depósitos.

A API mapeada com o *Spring Boot* serviu como ponte entre o solucionador e a aplicação, iniciando e retornando as soluções requisitadas pelo usuário e recebendo requisições do *front-end*. Para segurança e visualização dinâmica das soluções, foram adicionadas as bibliotecas de JWT e *Websocket*.

A biblioteca de JWT (JSON Web Token) foi utilizada para controle de acesso do usuário ao sistema. Assim quando é validada a entrada do usuário, a API através da biblioteca gera um *token* de acesso que é utilizado nas trocas de mensagens futuras, garantindo a segurança e sessão do usuário na aplicação.

Devido a possibilidade de assincronismo do solucionador, a biblioteca *Websocket* foi utilizada para o envio e retorno das soluções. Desta forma quando a aplicação envia um problema, ouve também sua solução para ser exibida no mapa, atualizando automaticamente as novas combinações geradas. Tal tecnologia necessitou de um estudo, para o entendimento do funcionamento assíncrono das chamadas entre o servidor e a interface de usuário. Desta maneira a aplicação consegue responder e exibir de forma dinâmica as soluções encontradas pelo solucionador, enriquecendo a experiência do usuário, do qual não tem que esperar até o final da execução para visualizar os resultados e progresso do solucionador.

3.2.2 Interface gráfica

A interface de usuário da aplicação web foi construída com o framework *React*, utilizando o *Redux* para o controle do fluxo de dados. Para a implementação do layout, utilizou-se componentes de layout da biblioteca *Material Design*.

A camada de interface de usuário mostrou algumas dificuldades, como a integração entre a biblioteca visual do *Google Maps*, processamento dos endereços de passageiros e fluxo de estados aplicados pelo *Redux*.

A abordagem para utilização do *Google Map,* não utilizou de bibliotecas de terceiros, incorporando a biblioteca dentro dos componentes do *React*, resultando em uma maior liberdade, controle dos layouts do mapa e maior complexidade de implementação.

O *redux* dispara e recebe as transações de dados entre a interface de usuário e a API. A Figura 7 exemplifica a conexão em funcionamento quando o usuário insere as informações incorretas cadastradas.

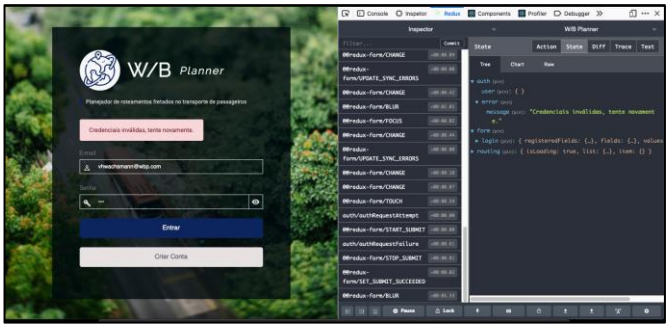

Figura 7: *Redux* mediando o fluxo de dados

Embora o fluxo de estados da biblioteca *Redux* agregue coesão e gerência unidirecional do fluxo de dados, necessita-se um entendimento do conceito, além de um aprofundamento para implementação e utilização dentro da mecânica do *React*. Tal abordagem impacta ao longo prazo dentro do projeto, com padronização, clareza e manutenção da aplicação. Entretanto a configuração, entendimento e aplicação das técnicas, aumentam consideravelmente o tempo de desenvolvimento. Cabendo ao programador controlar as camadas de abstração, em uma proporção condizente com o escopo e progresso do desenvolvimento. Desta forma, priorizou-se inicialmente o fluxo de autenticação para ser aplicado dentro do *Redux*.

Após a importação e sempre quando alterado o raio, o sistema processa e insere os pontos de embarque de acordo com o raio configurado, obedecendo a restrição de atendimento de passageiros e raio de locomoção. O algoritmo varre todos os passageiros inseridos, agrupando-os por proximidade. A cada iteração muda seu centro de busca de acordo com passageiros já adicionados, até percorrer todos, cobrindo assim os passageiros com pontos de embarques. Foram identificadas dificuldades no agrupamento de pontos próximos de acordo com o raio configurado. Sendo a distância entre pontos calculada linearmente, ao contrário das rotas que tiveram a utilização da mecânica de direções. Desta característica requisitou-se a possibilidade de arraste do ponto de embarque dentro do mapa, para ajustes precisos de localização, agregando assim com a realidade dos problemas enfrentados.

Figura 8: Inserção automática de pontos de embarque

Como pode ser visto na Figura 8, da qual o pin com a letra E exibe o ponto de embarque automaticamente inserido com seu raio de alcance, e o pin com a letra P os passageiros importados pelo usuário.

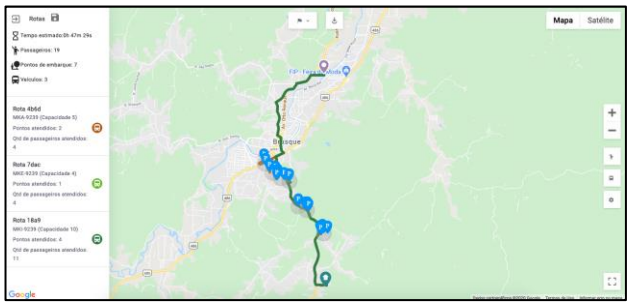

Figura 9: Solução de roteamento com rota e informações de otimização

A Figura 9 exibe a tela de roteamento com um problema configurado e executado dentro do solucionador. Podem-se observar os resultados retornados pelo solucionador, sendo exibidos no *card* com cada rota criada para os veículos inseridos, contendo o nome do veículo, capacidade, quantidade de pontos atendidos e a demanda de passageiros. Além do *card* é possível visualizar o percurso das rotas sugeridas no mapa, juntamente com as configurações inseridas como, passageiros, pontos gerados de acordo com o raio configurado, origem e destino dos veículos.

4 CONCLUSÕES

Um solucionador para roteamento de veículos foi desenvolvido, com foco no transporte fretado de passageiros. O estudo aplicou conceitos de programação de restrições na criação deste solucionador. A aplicação auxiliaria administradores de roteamentos na tarefa de gerenciar veículos, atendendo passageiros com limites locomotivos por meio de pontos de embarques.

A principal característica observada para a ferramenta foi o apoio e agilidade em montar os roteamentos, apontando gargalos de frotas e sugerindo configurações ao administrador de roteamentos. Assim ao configurar o roteamento da frota o utilizador obtém uma consultoria automática.

O retorno das rotas pelo solucionador se mostrou condizente com a realidade, embora observou-se que raros cenários possam apresentar uma desatualização de ruas do *Open Street Maps*. Sendo o *Google Maps* utilizado pela maioria

das empresas devido a atualidade dos mapas, necessitaria para tais cenários o tratamento em específico, ou incorporação do motor de direções do *Google Maps* como opção juntamente com o *GraphHopper*.

Constatou-se então uma real aplicação da ferramenta criada, tanto como contribuição para a área de pesquisa operacional, voltada para o roteamento de veículos fretados, como para o auxílio de administradores de roteamentos no transporte fretado de passageiros.

A continuidade da plataforma visto sua utilidade, abre uma gama de melhorias e adições. Pode-se citar como mais importantes, a exportação ou importação dos roteamentos para integração com outras ferramentas como o *My Maps* da Google. Desta forma, a ferramenta complementaria outras, já consolidadas e utilizadas, aumentando sua adesão dentro do mercado. A adição de *n* origens dentro do solucionador atenderia casos específicos de empresas que possuem depósitos estratégicos para execução de seus roteamentos, posicionando assim seus veículos em origens diferentes dentro de uma região.

Por fim, a complementação do processo de roteamento como visto, constitui-se em escalas de trabalho para motoristas, sendo conhecida como problema de janela de tempo. Utilizando esta pesquisa como base seria interessante a agregação do solucionador ao problema. Desta forma ao finalizar o roteamento, o planejador poderia adicioná-lo a um quadro de escalas, alocando motoristas de acordo com as restrições impostas. Agregando ao W/B Planner como um novo módulo, utilizando dos roteamentos criados, para gerenciar as frotas configuradas dentro de escalas de trabalho. Possibilitando aos utilizadores a gerência e flexibilidade das execuções dos roteamentos.

REFERENCES

[1] Caric, Tonci; Gold, Hrvoje. Vehicle Routing Problem. Viena, Áustria: In-Tech, 2008.
[2] Mauri, Geraldo. R; Lorena, Luiz A. Nogueira. Uma nova abordagem para o problema dial-a-ride. Produção, v. 19, 2009.
[3] Salles, Rosemberg S.; Neto, Gregório C. de M.; Cruz, Marta M. da C. An application of vehicle routing problem in chartered buses to transport employees using geographic information system, Federal University of Espírito Santo, Vitória, ES, Brasil, 2013.
[4] APT, Krzysztof R. Principles of Constraint Programming. Nova Iorque, EUA: Cambridge University Press, 2003.
[5] Wazlawick, Raul Sidenei. Metodologia de Pesquisa para Ciência da Computação. Rio de Janeiro, Brasio: Elsevier, 2009.
[6] The PostgreSQL Global Development Group. PostgreSQL. 1996-2020. Disponível em: <https://www.postgresql.org/about>. Acesso em 06 jul. 2020.
[8] Pivotal Software. Spring Framework. 2002-2020. Disponível em: <https://www.postgresql.org/about>. Acesso em: 04 jul. 2020.
[9] Red Hat Inc. OptaPlanner: An AI constraint solver. 2006-2020. Disponível em: <https://docs.optaplanner.org/7.29.0.Final/optaplanner-docs/pdf/index.pdf>. Acesso em: 16 nov. 2019
[10] Facebook Inc, React. 2020. Disponível em: <https://pt-br.reactjs.org/docs/getting-started.html>. Acesso em: 05 de jul. 2020.
[11] Abramov, Dan; Redux. 2015-2020. Disponível em: <https://redux.js.org/api/api-reference>. 05 de jul. 2020.
[12] Google LLC. OR-Tools. 2010-2020. Disponível em: <https://developers.google.com/optimization>. Acesso em: 7 abr. 2020.
[13] Yüceer, Ümit. An employee transport problem. Journal of industrial Engineering International, Mersi. Turquia, 2013.
[14] Casal, João Afonso Vieira. Vehicle Routing Problems: Investigação e construção de um Sistema de Informação Geográfica. 86 f. Dissertação (Mestrado) – Curso de Engenharia de Sistemas, Escola de Engenharia, Universidade do Minho, Minho, 2012.